公司shell脚本笔记

```
parameter='['`date +%Y-%m-%d' '%H:%M:%S,%3N`']'"[$$] 传入参数:$*"
```

parameter：表示当前时间，后面是时间格式

```
[ -f ~/apphome/aic_export_fps.sh ] || { echo "文件aic_export_fps.sh 不存在"
exit 1
}
```

- -f：校验常规文件是否存在，如果不存在则告知
- 错误退出

```
export org_id="${1}"
GROUP="$2"
export downOrUp=${GROUP%%_*}
export batch_date="$3"
```

设置脚本环境：传入各种参数

- org_id：id
- GROUP：组别
- batch_date：批处理时间

```
[ "${GROUP}" == 'input_xbCashStopList' -o "${GROUP}" == 'input_ecifSyn' ] && batch_date=`date -d "-1 day ago ${batch_date}" +%Y%m%d`
last_batch_date=`date -d "1 day ago ${batch_date}" +%Y%m%d`
next_batch_date=`date -d "-1 day ago ${batch_date}" +%Y%m%d`
```

如果(GROUP等于“input_xbCashStopList”**或者**等于“input_ecifSyn”)为真时：

批处理时间设置为一天后。

- 最后一次批处理时间为1天前
- 下一次批处理时间为1天后

```
export batch_date6=${batch_date:2}
export bussiness_date=`date +%Y%m%d`
LIST_FILE=$4
list_file_name=${LIST_FILE##*/}
export list_file_name=${list_file_name%%.*}
MD5_file=$5
export request_day=${6}
batch_number=$7
batch_no=$8
```

- 设置批处理时间为，截取2位后的时间
- 设置业务时间
- 传入列表文件
- 设置列表文件名
- 传入md5文件值
- 设置传入需求时间
- 传入批处理号
- 传入批处理编号

```
. $HOME/apphome/aic_export_fps.sh
[ $? != 0 ] && exit 99
. $HOME/apphome/sbin/fps/FPS_default.sh
```

- 导入fps脚本
- 函数运算结果不等于0，则告知错误退出
- 导入fps_default脚本

```
function log()
{
    echo '['`date +%Y-%m-%d' '%H:%M:%S,%3N`']'"[$$]""${file_id_log} ${step%%_*}" "$1"
}
```

log函数：

- 输出时间

```
#-------日间批PBOC--
if [ "${GROUP:0:16}" == 'output_PBOC10003' ]
then
    PBOC_out_filename=${MD5_file#*:}
    MD5_file=':'
fi
```

如果组别的0-16位为"output_PBOC10003",则：

- 设置文件名
- 设置md文件为：

```
"
[ "${default_sleepTime}" == '' ] && default_sleepTime=10
```

如果默认睡眠时间为空，则设置默认睡眠时间为10

```
. ${local_fps_function_path}/FPS_DOWN.sh
. ${local_fps_function_path}/FPS_DOWNOK.sh
. ${local_fps_function_path}/FPS_CKRSLT.sh
. ${local_fps_function_path}/FPS_ICCARD.sh
. ${local_fps_function_path}/FPS_UP.sh
. ${local_fps_function_path}/FPS_UNZIP.sh
. ${local_fps_function_path}/FPS_ZIP.sh
. ${local_fps_function_path}/FPS_CG.sh
. ${local_fps_function_path}/FPS_CK.sh
. ${local_fps_function_path}/FPS_RM.sh
. ${local_fps_function_path}/FPS_ADDHEAD.sh
. ${local_fps_function_path}/FPS_PBOC.sh
. ${local_fps_function_path}/FPS_CKIFMCZ.sh
. ${local_fps_function_path}/FPS_CK.sh
. ${local_fps_function_path}/FPS_MK.sh
. ${local_fps_function_path}/FPS_MG.sh
. ${local_fps_function_path}/FPS_MGETC.sh
. ${local_fps_function_path}/FPS_MGSHELL.sh
. ${local_fps_function_path}/FPS_CPPATH.sh
. ${local_fps_function_path}/FPS_FILES.sh
. ${local_fps_function_path}/FPS_RNAME.sh
. ${local_fps_function_path}/FPS_RNAIC.sh
. ${local_fps_function_path}/FPS_CHECK.sh
. ${local_fps_function_path}/FPS_SD.sh
```

导入一堆脚本文件

```
fps_type=${GROUP%%_*}
```

设置fps的类别为GROUP的变量“_”前的所有变量名

```
#--------------日志以及处理结果目录------------
LOG_FILE_PATH="${local_fps_log_path}/${org_id}/${batch_date}"
LOG_FILE="${LOG_FILE_PATH}/FPS_${list_file_name}_${org_id}_${batch_date}_${GROUP}.log"
RESULT_FILE_PATH="${local_fps_result_path}/${org_id}/${batch_date}"
RESULT_FILE="${RESULT_FILE_PATH}/FPS_${list_file_name}_${org_id}_${batch_date}_${GROUP}.result"
```

- 设置日志文件路径
- 设置日志文件名
- 设置结果文件路径
- 结果文件名

```
#[ "$7" == 'C' ] && rm ${RESULT_FILE}
[ -d ${LOG_FILE_PATH} ] || mkdir -p ${LOG_FILE_PATH}
[ -d ${RESULT_FILE_PATH} ] || mkdir -p ${RESULT_FILE_PATH}
[ -f ${LOG_FILE} ] || >${LOG_FILE}
[ -f ${RESULT_FILE} ] || >${RESULT_FILE}
```

- 检查LOG_FILE_PATH是否是目录，不是则创建
- 检查RESULT_FILE_PATH是否为目录，不是则创建
- 检查LOG_FILE是否为普通文件，不是则创建
- 检查RESULT_FILE是否为普通文件，不是则创建

```
#------所有输出重定向到日志文件-----------------
exec 1>>${LOG_FILE} 2>>${LOG_FILE}


[ -d ${MONITORING_PATH} ] || mkdir -p ${MONITORING_PATH}
[ -f ${MONITORING_FILE} ] || >${MONITORING_FILE}
```

- 输出1到日志文件中，输出2到日志文件中
- 如果MONITORING_PATH目录不存在则创建
- 如果MONITORING_FILE目录不存在则创建

```
#保存当前处理文件的文件名

#---------记录脚本开始执行时间到日志文件中----------
echo "${parameter}" >>${LOG_FILE}
[ ${#org_id} -ne 12 ] && write_result null 1 "机构号检查"
[ "${fps_type}" != 'input' -a "${fps_type}" != 'output' ] && write_result null 2 "group检查"
```

- 把元素追加到日志文件中
- 如果编号不等于12，则把write_result设置为null？？？
- 如果ftp的类型不是input并且类型也不是output？则？？？

```
FILE_IDS=(`echo "${FILE_ID_LIST}" |awk '{print $2}' |sort -u`)
```

- 输出文件id列表，将其重复项删除后复制给FILE_IDS

```
#echo "FILE_IDS=${FILE_IDS[@]}"
[ "${default_inputNextFile}" != '' -a "${default_inputNextFile//[0-9]*:[0-9]*/}" == '' ] || default_inputNextFile=''
[ "${default_outputNextFile}" != '' -a "${default_outputNextFile//[0-9]*:[0-9]*/}" == '' ] || default_outputNextFile=''
[ "${default_inputNextFile}" != '' -a "${default_outputNextFile}" == '' ] && default_outputNextFile=${default_inputNextFi
[ "${default_inputNextFile}" == '' -a "${default_outputNextFile}" != '' ] && default_inputNextFile=${default_outputNextFi
[ "${default_inputNextFile}" == '' -a "${default_outputNextFile}" == '' ] && d_NextFile=''
[ "${default_inputNextFile}" != '' -a "${default_outputNextFile}" != '' ] && d_NextFile='Y'

[ "${fps_type}" == 'input' ] && temp=(${default_inputNextFile//:/ })
[ "${fps_type}" == 'output' ] && temp=(${default_outputNextFile//:/ })
```

- （如果默认的输入和输出的下一文件不为空并且前9位不为空）为假，则设置这个变量为空。
- （如果默认的输入和输出的下一文件不为空并且前9位不为空）为真，则设置输出文件为输入文件变量/设置输入文件为输出文件的名。
- （如果默认的输入和输出的下一文件为空并且前9位为空）为真，则设置d_NextFile为空
- （如果默认的输入和输出的下一文件为空并且前9位不为空）为真，则设置d_NextFile变量为y

```
[ "${fps_type}" == 'input' ] && temp=(${default_inputNextFile//:/ })
[ "${fps_type}" == 'output' ] && temp=(${default_outputNextFile//:/ })
default_NextFile_sleepTime=${temp[0]}
default_NextFile_num=${temp[1]}
```

- 如果fps的类型为input为真，则设置temp为默认的下个文件的input地址替换为截取字符串
- 如果fps的类型为output为真，则设置temp为默认的下个文件的output地址替换为截取字符串
- 默认下一文件休眠时间为temp的数组一号元素
- 默认下一文件号为temp的数组二号元素

```
n_FILE_IDS=''
for file_id in ${FILE_IDS[@]}
do
    FILE_SETUP_LIST=`cat ${LIST_FILE} |grep "^${GROUP} " |awk '{if("'${file_id}'" == $3)print;}'`
    [ "${FILE_SETUP_LIST}" != '' ] && n_FILE_IDS="${n_FILE_IDS} ${file_id}"
done
FILE_IDS=(${n_FILE_IDS})
```

- 初始化文件id为空数组
- 循环文件id，读取文件LIST_FILE，搜索组名，将其查到结果进行判断，如果文件id等于传入参数，则输出出来。
- 设置文件ids为n_FILE_IDS

接下去是一个很长的无限循环方法，退出用的break，以下是对代码的解析。

```
while [ 1 ]
do
FINISH_FLAG_failed=''          初始化失败标记
FINISH_FLAG_wait=''            等待标记
FINISH_FLAG_successful=''      成功标记
for file_id in ${FILE_IDS[@]}  从文件的id数组开始进行第二个遍历
do
    result=0                   初始化结果
    file_id_log=${file_id}     给文件日志id赋值上文件id
    write_file_id=${file_id}   写入文件id赋值给上文件id
    eval FLAG_${file_id}=''    执行前扫描两次，初始化文件id标记
    [ "${GROUP:0:16}" == 'output_PBOC10003' ] && write_file_id="${PBOC_out_filename}.txt_${file_id}"   如果组别的0-16的字符串为“output”则将写入文件id赋值为----

    #cat ${LIST_FILE} |grep "^${GROUP}" |awk '{if("'${file_id}'" == $3)print;}' >${FILE_SETUP_LIST}
    #cat "${FILE_ID_LIST}" |awk '{if("'${file_id}'" == $2)print;}' >${FILE_LIST_2}
    FILE_SETUP_LIST=`cat ${LIST_FILE} |grep "^${GROUP} " |awk '{if("'${file_id}'" == $3)print;}'`   读取List_FILE文件，搜索组名匹配的文件将其打印出来并且赋值给FILE_SETUP_LIST
    FILE_LIST_2=`echo "${FILE_ID_LIST}" |awk '{if("'${file_id}'" == $2)print;}'`   读取FILE_ID_LIST变量，如果满足文件id等于给定的变量，则打印出来并且赋值给FILE_LIST_2

    [ "${FILE_SETUP_LIST}" == '' ] && continue   如果FILE_SETUP_LIST为空，则退出for循环，继续while循环
#echo "FILE_SETUP_LIST=${FILE_SETUP_LIST}"
#echo "FILE_LIST_2=${FILE_LIST_2}"

    SERVER_NUM=`echo "${FILE_LIST_2}" |wc -l`   读取FILE_LIST_2的变量并获取长度赋值给SERVER_NUM
    FILE_NAME_MG='' 初始化FILE_NAME_MG
    eval local_path_mg="${local_fps_base_file_path}/${list_file_name}/${file_id}/FILES_MG"  扫描并赋值mg给予地址
    [ -d ${local_path_mg} ] || mkdir -p ${local_path_mg}   如果path_mg这个目录不存在，则创建这个目录

    eval local_path="${local_fps_base_file_path}/${list_file_name}/${file_id}"   扫描local_path，并给地址赋值
```

```
[ -d ${local_path_mg} ] || mkdir -p ${local_path_mg}   检索文件目录path_mg是否存在，不在就创建文件目录

eval local_path="${local_fps_base_file_path}/${list_file_name}/${file_id}"   扫描并且赋值local_path
[ -d ${local_path} ] || mkdir -p ${local_path}   检索文件目录local_mg是否存在，不在就创建文件目录

rs=`grep "^${GROUP},${write_file_id}," ${RESULT_FILE} |awk -F, '{print $3}'`   检索符合条件的打印并赋值给rs
if [ "${rs}" == '成功' ]   如果rs为成功，则进行以下操作
then
    log "${write_file_id}已处理过，跳过此文件"   打印日志
    eval FLAG_${file_id}='successful'   给成功标记赋值成功
    continue   退出当前for循环
fi
[ "${MD5_file#*:}" != '' ] && write_file_id="${MD5_file#*:}_${file_id}"   如果md5值不为空，则对写入文件id进行赋值
#[ -d ${local_path_mg} ] && rmFile "PATH_FILE" "${local_path_mg}"
[ "${MD5_file#*:}" == '' ] && rmFile "PATH_FILE" "${local_path}"   如果md5值文件为空，则执行删除文件方法
[ -d ${local_path_mg} ] || mkdir -p ${local_path_mg}   扫描local_path_mg目录是否存在不在就创建
cd ${local_path}   转到该目录下

SERVER_NUM_NOW=0
while read SERVER_NAME  FILE_ID  PARAMS   读取写入的三个元素
do
    #echo "SERVER_NAME=${SERVER_NAME}  FILE_ID=${FILE_ID}  PARAMS=${PARAMS}"
    SERVER_NUM_NOW=$((${SERVER_NUM_NOW} + 1))   当前数字+1
    #--------获取文件个性化参数-------------------------
    for n in ${PARAMS}   用传入的元素进行第三次循环
    do
        [ ${n:0:2} != '--' ] && continue   如果前两个字符为--则退出第三层循环
        eval ${SERVER_NAME}_${FILE_ID}_${n#--}   扫描我们传入的两个参数的组合参数
    done
    eval param_matchDate=\${${SERVER_NAME}_${FILE_ID}_matchDate}   扫描param_matchDate并且给予赋值
```

```
        param_matchDate_result=`echo ${batch_date} |grep "${param_matchDate}"`
        if [ "${param_matchDate}" != '' -a "${param_matchDate_result}" = "" ]
        then
            log "仅在符合以下规则的日期处理此文件：${param_matchDate} 当前日期${batch_date}不符合规则，跳过处理此文件"
            continue
        fi

        [ "${MD5_file#*:}" != '' ] && eval ${SERVER_NAME}_${FILE_ID}_fileName\=${MD5_file#*:}

#echo "SERVER_NAME=${SERVER_NAME}"
        DEFAULT_PARAMS="`env |grep '^default_' | sed 's/^default_/--/g'`"
        eval SERVER_PARAMS="\${${SERVER_NAME}}"
        ALL_PARAMS=''
        for n in ${PARAMS} ${DEFAULT_PARAMS} ${SERVER_PARAMS}
        do
            [ ${n:0:2} != '--' ] && continue
            ALL_PARAMS="${ALL_PARAMS} ${n%%=*}"
        done

        for value in ${ALL_PARAMS}
        do
            getParamValue "${value#--}"
            eval export ${value#--}\=\${getParamValue_value}
        done
```

查读取batch_date，查找param_matchDate并把它赋值给
如果param_matchDate不为空并且result为空
直接将其打印出来并退出第三个循环体
如果md5文件不为空，则扫描并赋值filename
查找‘default’字符串，将其替换成xxx--/g，再赋值给默认元素
扫描并替换元素
对传入的元素进行循环遍历
如果传入的前两位为--则退出当前循环
设置所有元素，为所有元素加上进行操作的n（删除第一个=号及其右边字符串）
对所有元素进行循环
获取元素的值
扫描并且设置参数的值

```
#-------获取文件处理规则并处理文件------------------ 对读取的所有文件进行循环
while read LIST_group LIST_type LIST_FILE_L  LIST_DESC LIST_p11 LIST_p12 LIST_p13 LIST_p14  LIST_d_host LIST_d_path LIST_d_file LIST_l_path1
do
    eval R_createEmptyFile=\${${SERVER_NAME}_${FILE_ID}_createEmptyFile}
    R_createEmptyFile=(${R_createEmptyFile//:/ })
    R_createEmptyFile_type=${R_createEmptyFile[0]}
    R_createEmptyFile_value=${R_createEmptyFile[1]}
    [ "${R_createEmptyFile_value}" == '' ] && R_createEmptyFile_value=1
    eval R_up_fileName="${LIST_u_file}"

    if [ "${downOrUp}" == 'input' -a "${LIST_d_host}" == '-' ]
    then
        eval DOWN_ip=\${${SERVER_NAME}_ip}
        eval DOWN_user=\${${SERVER_NAME}_user}
        eval DOWN_path="\${${SERVER_NAME}_path}${LIST_d_path%/}"
        eval DOWN_base64Pwd=\${${SERVER_NAME}_base64Pwd}
        eval DOWN_javaPwd=\${${SERVER_NAME}_javaPwd}
        eval DOWN_pwd=\${${SERVER_NAME}_pwd}
        eval DOWN_transferType=\${${SERVER_NAME}_transferType}
        eval DOWN_port=\${${SERVER_NAME}_port}
    else
        eval DOWN_ip="\${${LIST_d_host}_ip}"
        eval DOWN_user="\${${LIST_d_host}_user}"
        eval DOWN_path="\${${LIST_d_host}_path}${LIST_d_path%/}"
        eval DOWN_base64Pwd="\${${LIST_d_host}_base64Pwd}"
        eval DOWN_javaPwd="\${${LIST_d_host}_javaPwd}"
        eval DOWN_pwd="\${${LIST_d_host}_pwd}"
        eval DOWN_transferType="\${${LIST_d_host}_transferType}"
        eval DOWN_port="\${${LIST_d_host}_port}"
    fi
```

扫描并且给以下所有元素赋值
给予类型赋值
给予文件值赋值
扫描并赋值
如果参数downOrUp等于input并且LIST_d_host为-，则进行一系列赋值

```
  if [ "${downOrUp}" == 'output' -a "${LIST_u_host}" == '-' ]
  then
      eval UP_ip=\${${SERVER_NAME}_ip}
      eval UP_user=\${${SERVER_NAME}_user}
      eval UP_path=\${${SERVER_NAME}_path}
      eval UP_base64Pwd=\${${SERVER_NAME}_base64Pwd}
      eval UP_javaPwd=\${${SERVER_NAME}_javaPwd}
      eval UP_pwd=\${${SERVER_NAME}_pwd}
      eval UP_transferType=\${${SERVER_NAME}_transferType}
      eval UP_port=\${${SERVER_NAME}_port}
  else
      eval UP_ip="\${${LIST_u_host}_ip}"
      eval UP_user="\${${LIST_u_host}_user}"
      eval UP_path="\${${LIST_u_host}_path}${LIST_u_path%/}"
      eval UP_base64Pwd="\${${LIST_u_host}_base64Pwd}"
      eval UP_javaPwd="\${${LIST_u_host}_javaPwd}"
      eval UP_pwd="\${${LIST_u_host}_pwd}"
      eval UP_transferType="\${${LIST_u_host}_transferType}"
      eval UP_port="\${${LIST_u_host}_port}"
  fi
  eval DOWN_file="${LIST_d_file}"
  DOWN_empty_flag=''
```

如果downOrUp为output 并且LIST_u_host等于-，则进行多次扫描赋值
扫描赋值完毕后给予DOWN_file赋上LIST_d_file
初始化标记

```
if [ "${downOrUp}" == 'input' -o ${SERVER_NUM_NOW} -eq 1 ]
then
    if [ "${R_createEmptyFile_type}" == 'Y' ]
    then
        >${local_path_mg}/${R_up_fileName}
        FILE_NAME_MG=${R_up_fileName}
    else
        #echo "${LIST_p11}"
        steps1=("${LIST_p11//\|/ }")
        for step in ${steps1[@]}
        do
            eval file_id_flag=\${FLAG_${file_id}}
            [ "${d_NextFile}" == 'Y' ] && [ "${file_id_flag}" == 'wait' -o "${file_id_flag}" == 'failed' ] && break
            eval step=\"${step}\"
            step_name=${step%%_*}
            step_type=${step#*_}
            [ "${step_name}" == "${step_type}" ] && step_type=''
        #   echo ${step_name} ${step_type}
            [ "${DOWN_empty_flag}" == 'Y' -a "${step_name}" != 'RNAME' ] && continue
            ${step_name} ${step_type}
        done
        step=''

        #echo "${LIST_p12}"
        steps2=("${LIST_p12//\|/ }")
        for step in ${steps2[@]}
        do
            eval file_id_flag=\${FLAG_${file_id}}
            [ "${d_NextFile}" == 'Y' ] && [ "${file_id_flag}" == 'wait' -o "${file_id_flag}" == 'failed' ] && break
            eval step=\"${step}\"
            step_name=${step%%_*}
            step_type=${step#*_}
            [ "${step_name}" == "${step_type}" ] && step_type=''
            ${step_name} ${step_type}
        done
```

如果downOrUp等于input或者当前数字为1，则进入第一层if判断体
如果类别为Y则执行以下操作
追加path_mg/up_filename
给予FILE_NAME_MG赋值
用正则表达式给steps1赋值
用step进行一层循环
扫描并且给予文件id标记赋值
如果下一个文件为y并且（文件的标记为成功或者失败）则退出全部循环
给予步骤赋值
给予步骤名赋值（截取_后的字符串）
给予步骤类别赋值
如果步骤名和步骤类别相等，则赋值步骤类别为空
如果文件标记为y或者步骤名为RNAME则退出该层循环体
初始化step
给予step2赋值
使用step2进行循环
扫描并给予file_id_flag赋值
如果下一文件为Y并且（标记为等待或者失败）则退出全部循环
扫描并赋值step
截取_后的元素为step_name赋值
位step的类型赋值
如果名字和类型相同，则设置类型为空

这里不是为空，是设置为名字+类型。

```
[ "${d_NextFile}" == 'Y' ] && [ "${file_id_flag}" == 'wait' -o "${file_id_flag}" == 'failed' ] && break
```

这里是如果文件标记为Y，则判断标记是否为wait或者failed，是则退出。

```
            #echo "${LIST_p13}"
            steps3=("${LIST_p13//\|/ }")
            for step in ${steps3[@]}        对step3数组进行循环
            do
                eval file_id_flag=\${FLAG_${file_id}}   扫描并赋值id的标志位
                  [ "${d_NextFile}" == 'Y' ] && [ "${file_id_flag}" == 'wait' -o "${file_id_flag}" == 'failed' ] && break
                eval step=\"${step}\"   扫描并赋值
                step_name=${step%%_*}   赋值名
                step_type=${step#*_}    赋值类型
                [ "${step_name}" == "${step_type}" ] && step_type=''   如果文件名等于类别，则将类别设置为名字+类别
                ${step_name} ${step_type}
            done
            step=''     执行完毕重新初始化step
        fi
    fi
```

如果下一文件标记为Y则（判断文件标记是否为wait或faiiled），是则退出所有循环体

```
    if [ ${downOrUp} == 'output' -o ${SERVER_NUM_NOW} -eq ${SERVER_NUM} ]   如果downOrUp为output或者两个标记相等
    then
        #echo "${LIST_p14}"
        steps4=("${LIST_p14//\|/ }")    获取step4
        for step in ${steps4[@]}        用这个数组进行循环
        do
            eval file_id_flag=\${FLAG_${file_id}}扫描并初始化文件id
            [ "${d_NextFile}" == 'Y' ] && [ "${file_id_flag}" == 'wait' -o "${file_id_flag}" == 'failed' ] && break
            eval step=\"${step}\"
            step_name=${step%%_*}   赋值步骤名
            step_type=${step#*_}    赋值类别
            [ "${step_name}" == "${step_type}" ] && step_type=''   如果类别名和步骤名相等，则赋值类别为名字+类别名
            ${step_name} ${step_type}
        done
        step=''     初始化步骤
    fi
    eval file_id_flag=\${FLAG_${file_id}}   扫描并赋值标记
    [ "${d_NextFile}" == 'Y' ] && [ "${file_id_flag}" == 'wait' -o "${file_id_flag}" == 'failed' ] && break
done <<EOF      将SETUP_LIST读入
${SETUP_LIST}
```

如果下一文件标识为Y，则检查是否为wait或者fail，是则退出全部循环体

如果下一文件标识为Y，则检查是否为wait或者fail，是则退出全部循环体

```
        eval file_id_flag=\${FLAG_${file_id}}   扫描并且设置标记
        [ "${d_NextFile}" == 'Y' ] && [ "${file_id_flag}" == 'wait' -o "${file_id_flag}" == 'failed' ] && break
    done <<EOF      如果下一个文件标记为y则判断（文件标记是否为wait或者failed），是则退出
${FILE_LIST_2}      读取文件FILE_LIST_2
EOF
    eval file_id_flag=\${FLAG_${file_id}}   如果下一个文件标记为y则判断（文件标记是否为wait或者failed），是则退出
    [ "${d_NextFile}" == 'Y' ] && [ "${file_id_flag}" == 'wait' -o "${file_id_flag}" == 'failed' ] && continue
    write_result "${FILE_NAME}" 0 "处理完成了"
done
[ "${d_NextFile}" != 'Y' ] && break     判断下一文件标记不等于Y则不执行，退出
if [ ${count_num} -ge ${default_NextFile_num} ]  如果统计数大于等于默认下一文件统计数
then
    log "第${count_num}/${default_NextFile_num}次轮询处理文件，轮询等待次数过多，退出脚本"
    exit 1          则打印日志，报错
fi

FILE_ID=''          初始化文件id，文件日志id，步骤
file_id_log=''
step=''
for file_id in ${FILE_IDS[@]}   对文件id进行循环
do
    eval file_id_flag=\${FLAG_${file_id}}   给标记赋值
    [ "${file_id_flag}" == 'failed' ] && FINISH_FLAG_failed='Y'  如果文件id标记为失败，则赋值失败标记为Y
    [ "${file_id_flag}" == 'wait' ] && FINISH_FLAG_wait='Y'      如果文件id标记为wait，则赋值等待标记为Y
    [ "${file_id_flag}" == 'successful' ] && FINISH_FLAG_successful='Y'
      result_desc=`printf "文件:%-30s 状态:%-s" "${file_id}" "${file_id_flag}"`   如果为成功，则标记成功标记为Y，
    log "${result_desc}"                                                        打印结果描述，包括文件名字30个字符，
done                                                                            状态一个字符
if [ "${FINISH_FLAG_failed}" == 'Y' ]       如果失败标记则打印日志并且报错
then
    log "存在失败的处理，请查看失败原因"
    exit 1
else
    if [ "${FINISH_FLAG_wait}" != 'Y' ]     成功则成功退出
    then
        log "文件处理已完成"
        exit 0
    fi
fi
```